FOUNDATION-0.11 PART 2: Grafana Dashboard Setup

OptiInfra Development Series

Phase: Foundation (Week 1)

Component: Monitoring Infrastructure - Grafana

Estimated Time: 20 minutes setup + 10 minutes validation

Dependencies: P-01 (Bootstrap), 0.10 (Shared Utilities), 0.11-PART1 (Prometheus)

📋 Overview

This prompt creates comprehensive Grafana dashboards for OptiInfra, including:

- Grafana server with Prometheus data source
- 7 pre-built dashboards (Overview + 4 agents + Infrastructure + Alerts)
- Real-time metrics visualization
- Alert integration and notifications
- Dashboard JSON exports for version control

🎯 Objectives

By the end of this prompt, you will have:

1. ✅ Grafana server running with Prometheus data source configured
2. ✅ 7 comprehensive dashboards monitoring all system aspects
3. ✅ Real-time metrics updating every 5-15 seconds
4. ✅ Alert visualization and notification channels
5. ✅ Dashboard provisioning for automated deployment
6. ✅ Export-ready dashboard configurations

🔧 Prerequisites

Before starting, ensure:

- [ ] FOUNDATION-0.11 PART 1 completed - Prometheus running and scraping

- [ ] All services exposing metrics on /metrics endpoints
- [ ] Port 3000 available for Grafana
- [ ] Prometheus accessible at http://prometheus:9090

📝 Detailed Windsurf Prompt

Create a complete Grafana monitoring dashboard suite for OptiInfra multi-agent system.

CONTEXT:
- Multi-agent LLM infrastructure optimization platform
- 4 agents (Cost, Performance, Resource, Application) + 1 orchestrator
- Prometheus collecting metrics from all services
- Need comprehensive, real-time visualization
- Project root: /optiinfra

REQUIREMENTS:

1. GRAFANA SETUP (monitoring/grafana/)
   Create complete Grafana configuration:
   - grafana.ini: custom config with anonymous auth disabled
   - provisioning/datasources/prometheus.yml: Prometheus data source
   - provisioning/dashboards/optiinfra.yml: dashboard provider
   - provisioning/notifiers/: notification channels (Slack placeholder)

2. DASHBOARD 1: SYSTEM OVERVIEW (dashboards/overview.json)
   Single-pane-of-glass for entire OptiInfra:
   - Top row: Key metrics
     * Total cost savings (today, this week, this month)
     * Active optimizations count
     * System health (all services up/down count)
     * Average quality score
   - Second row: Service status
     * Panel for each service (orchestrator + 4 agents)
     * Green if up, red if down
     * Request rate (QPS)
     * Error rate (%)
     * P95 latency
   - Third row: Trends (time series)
     * Cost savings over time
     * Request rate by agent
     * Error rate by agent
   - Bottom row: Recent alerts

* Active alerts table
* Alert history graph

3. DASHBOARD 2: COST AGENT (dashboards/cost-agent.json)
Deep dive into cost optimization:
- Top metrics row:
* Total savings (counter)
* Savings rate (per hour)
* Active recommendations
* Recommendation confidence (avg)
- Cost breakdown:
* Savings by optimization type (spot, RI, right-sizing) - pie chart
* Savings over time by type - stacked area chart
- Execution metrics:
* Execution success rate
* Execution duration (P50, P95, P99)
* Failed executions (last 24h)
- Cloud provider breakdown:
* Cost by provider (AWS, GCP, Azure) - bar chart
* Utilization by provider - gauge
- LLM usage:
* API calls by provider/model
* Token usage
* LLM cost tracking
- Recent recommendations table:
* Type, confidence, estimated savings, status

4. DASHBOARD 3: PERFORMANCE AGENT (dashboards/performance-agent.json)
Performance optimization metrics:
- Top metrics:
* Average latency improvement ratio
* Current throughput (QPS)
* KV cache hit rate
* Active optimizations
- Latency tracking:
* P50/P95/P99 latency over time
* Latency by optimization type
* Before/after comparison chart
- Throughput metrics:
* Requests per second
* Tokens per second
* Batch size distribution
- KV cache performance:
* Hit rate over time

* Cache size
* Eviction rate
- Quantization impact:
* Speedup by quantization level (FP16, FP8, INT8)
* Accuracy impact (if tracked)
- vLLM/TGI/SGLang metrics:
* Engine-specific metrics
* Model load time
* GPU memory usage

5. DASHBOARD 4: RESOURCE AGENT (dashboards/resource-agent.json)
Resource utilization and scaling:
- Top metrics:
* Average GPU utilization
* Active GPUs
* Scaling events (last hour)
* Resource consolidation ratio
- GPU metrics:
* Utilization by GPU ID - heatmap
* Memory usage by GPU - stacked bar
* Temperature by GPU - gauge
* Power consumption
- CPU/Memory:
* CPU utilization by service
* Memory usage by service
* Disk I/O
- Scaling events:
* Scale up/down events over time
* Auto-scaling triggers
* Scaling duration
- KVOptkit integration:
* KV cache optimization events
* Memory saved
- Cost impact:
* Resource cost by type
* Idle resource cost

6. DASHBOARD 5: APPLICATION AGENT (dashboards/application-agent.json)
Quality monitoring and validation:
- Top metrics:
* Current quality score (avg)
* Regression detections (24h)
* Active A/B tests
* Auto-rollback events

- Quality scores over time:
  * Relevance score
  * Coherence score
  * Hallucination rate
  * Overall quality score
- Regression detection:
  * Regressions by type
  * Detection latency
  * False positive rate
- A/B testing:
  * Test results by variant
  * Statistical significance
  * Winning variants
- Validation metrics:
  * Validation duration
  * Approval/rejection rate
  * Rollback triggers
- Quality by optimization:
  * Quality impact of cost optimizations
  * Quality impact of performance optimizations
  * Quality safety margin

7. DASHBOARD 6: INFRASTRUCTURE (dashboards/infrastructure.json)
   Database and infrastructure health:
- Database metrics:
  * PostgreSQL: connections, queries/sec, cache hit rate
  * ClickHouse: queries/sec, inserted rows, compression ratio
  * Qdrant: vectors stored, search latency, index size
  * Redis: memory usage, keys, hit rate
- Orchestrator metrics:
  * Request routing (by agent)
  * Coordination conflicts
  * Approval workflow duration
  * Agent health status
- Network metrics:
  * Request latency by service pair
  * Data transfer rates
  * Connection errors
- Container metrics:
  * CPU usage by container
  * Memory usage by container
  * Restart count
  * Container health

8. DASHBOARD 7: ALERTS (dashboards/alerts.json)

Alert monitoring and history:

- Active alerts:
  * List of firing alerts
  * Alert severity
  * Time since firing
- Alert history:
  * Alerts over time by severity
  * Most frequent alerts
  * Mean time to resolution
- Alert grouping:
  * By service
  * By agent
  * By severity
- Alert rules status:
  * Rule evaluation duration
  * Rule evaluation errors
- Notification status:
  * Notifications sent
  * Notification failures

9. DOCKER COMPOSE UPDATES (docker-compose.yml)

Add Grafana service:

- Image: grafana/grafana:latest
- Volumes:
  * ./monitoring/grafana/grafana.ini:/etc/grafana/grafana.ini
  * ./monitoring/grafana/provisioning:/etc/grafana/provisioning
  * ./monitoring/grafana/dashboards:/var/lib/grafana/dashboards
  * grafana-storage:/var/lib/grafana
- Ports: 3000:3000
- Environment:
  * GF_SECURITY_ADMIN_PASSWORD=optiinfra_admin
  * GF_USERS_ALLOW_SIGN_UP=false
  * GF_SERVER_ROOT_URL=http://localhost:3000
- Depends_on: prometheus
- Networks: optiinfra-network

10. DASHBOARD TEMPLATES & VARIABLES

Each dashboard should use:

- Time range selector (last 5m, 15m, 1h, 6h, 24h, 7d)
- Refresh rate: 5s, 15s, 30s, 1m, 5m
- Variables:
  * $datasource (Prometheus)
  * $service (orchestrator, cost-agent, etc.)

* $agent (for agent-specific panels)
* $interval (auto-calculated based on time range)
- Consistent color scheme:
* Green: #00C853 (success, savings, improvements)
* Red: #F44336 (errors, problems, alerts)
* Orange: #FF9800 (warnings, degraded)
* Blue: #2196F3 (info, neutral metrics)

11. TESTING & VALIDATION (monitoring/tests/test_grafana.py)
Create tests:
- Test Grafana API accessible
- Test Prometheus data source connected
- Test all 7 dashboards loaded
- Test dashboard panels have data
- Test variables resolve correctly
- Test alerts are visible

TECHNICAL SPECIFICATIONS:
- Grafana version: 10.2+ (latest)
- Dashboard schema version: 36+
- Panel types: Graph, Stat, Gauge, Bar chart, Pie chart, Table, Heatmap
- Query language: PromQL
- Refresh intervals: 5s minimum for real-time panels
- Time ranges: Use relative ranges (now-5m to now)
- Units: Use appropriate units (percent, bytes, seconds, requests/sec)

FILE STRUCTURE:
```
monitoring/
├── grafana/
│   ├── grafana.ini            # Grafana config
│   ├── provisioning/
│   │   ├── datasources/
│   │   │   └── prometheus.yml   # Prometheus datasource
│   │   ├── dashboards/
│   │   │   └── optiinfra.yml    # Dashboard provider
│   │   └── notifiers/
│   │       └── slack.yml        # Notification channels
│   └── dashboards/
│       ├── overview.json
│       ├── cost-agent.json
│       ├── performance-agent.json
│       ├── resource-agent.json
│       ├── application-agent.json
│       ├── infrastructure.json
```

```
│       └── alerts.json
├── tests/
│   └── test_grafana.py
└── README-grafana.md
```

DASHBOARD DESIGN PRINCIPLES:

1. Most important metrics at the top
2. Color coding: Green=good, Red=bad, Orange=warning
3. Use appropriate visualizations:
   - Counters/totals: Stat panel with sparkline
   - Rates: Graph (time series)
   - Current state: Gauge
   - Distributions: Heatmap or histogram
   - Comparisons: Bar chart
   - Proportions: Pie chart
   - Lists: Table
4. Include drill-down links between dashboards
5. Add helpful descriptions to panels
6. Use consistent naming conventions
7. Group related panels in rows
8. Make heavy use of variables for flexibility

PROMQL QUERY EXAMPLES:

Cost Savings:
- Total savings: sum(increase(cost_savings_total[24h]))
- Savings rate: rate(cost_savings_total[5m]) * 3600
- By type: sum by (type) (increase(cost_savings_total[24h]))

Performance:
- P95 latency: histogram_quantile(0.95, rate(request_duration_seconds_bucket[5m]))
- Request rate: sum(rate(requests_total[1m])) by (service)
- Error rate: sum(rate(errors_total[5m])) / sum(rate(requests_total[5m]))

Resource:
- GPU utilization: avg(gpu_utilization) by (gpu_id)
- Scaling events: increase(scaling_events_total[1h])

Quality:
- Quality score: avg(quality_score)
- Regressions: increase(regression_detections_total[24h])

IMPLEMENTATION NOTES:
1. Start with grafana.ini and provisioning configs

2. Create simple overview dashboard first
3. Test data source connection
4. Build agent-specific dashboards
5. Add infrastructure and alerts dashboards
6. Test all panels have data
7. Export dashboard JSONs
8. Update docker-compose.yml
9. Test provisioning (dashboards auto-load)

BEST PRACTICES:
- Use template variables to make dashboards reusable
- Add panel descriptions (info icon)
- Use links between related dashboards
- Set appropriate time ranges per panel
- Use repeat panels for multi-instance metrics
- Add thresholds to gauges and stat panels
- Use overrides for series-specific colors
- Enable legend with values (min, max, avg, current)

VALIDATION STEPS:
After implementation, verify:
1. Open http://localhost:3000
2. Login with admin/optiinfra_admin
3. Check "Home" - should see 7 OptiInfra dashboards
4. Open each dashboard - all panels load with data
5. Test time range selector - data updates
6. Test refresh - data updates automatically
7. Test variables - dropdowns work
8. Check Prometheus data source in Settings

Generate complete, production-ready code with:
- Complete grafana.ini configuration
- All provisioning YAML files
- All 7 dashboard JSON files with proper structure
- Docker Compose updates
- Comprehensive tests
- Detailed README with screenshots descriptions

✅ Success Criteria

After completing this prompt, verify:

1. Grafana Access

```bash
# Check Grafana is running
curl http://localhost:3000/api/health
# Expected: {"database": "ok", "version": "10.2.x"}

# Login test
curl -u admin:optiinfra_admin http://localhost:3000/api/datasources
# Expected: JSON array with Prometheus datasource
```

2. Data Source Connection

```bash
# Test Prometheus datasource
curl -u admin:optiinfra_admin \
  http://localhost:3000/api/datasources/proxy/1/api/v1/query?query=up
# Expected: Query results from Prometheus
```

3. Dashboards Loaded

```bash
# List all dashboards
curl -u admin:optiinfra_admin http://localhost:3000/api/search | jq '.[].title'
# Expected:
# "OptiInfra Overview"
# "Cost Agent"
# "Performance Agent"
# "Resource Agent"
# "Application Agent"
# "Infrastructure"
# "Alerts"
```

4. Dashboard Data Validation

Open each dashboard in browser and verify:

Overview Dashboard (http://localhost:3000/d/optiinfra-overview)

- [ ] Total cost savings shows a number (even if 0)
- [ ] All 5 service status panels show green or red

- [ ] Request rate graph shows time series
- [ ] Error rate graph shows data
- [ ] Alerts table loads (may be empty)

Cost Agent Dashboard (http://localhost:3000/d/optiinfra-cost)

- [ ] Savings metrics show values
- [ ] Savings by type pie chart renders
- [ ] Savings over time graph shows trend
- [ ] Recommendations table has columns
- [ ] LLM usage panels show data

Performance Agent Dashboard (http://localhost:3000/d/optiinfra-performance)

- [ ] Latency improvement ratio shows value
- [ ] P50/P95/P99 latency graphs render
- [ ] Throughput QPS shows number
- [ ] KV cache hit rate displays

Resource Agent Dashboard (http://localhost:3000/d/optiinfra-resource)

- [ ] GPU utilization heatmap renders
- [ ] Scaling events graph shows data
- [ ] CPU/Memory charts display
- [ ] Resource cost panels show values

Application Agent Dashboard (http://localhost:3000/d/optiinfra-application)

- [ ] Quality score gauge shows value
- [ ] Regression detection counter displays
- [ ] Quality scores over time graph renders
- [ ] A/B test panels show data

Infrastructure Dashboard (http://localhost:3000/d/optiinfra-infra)

- [ ] All 4 database sections show metrics
- [ ] Orchestrator metrics display
- [ ] Container metrics render

Alerts Dashboard (http://localhost:3000/d/optiinfra-alerts)

- [ ] Active alerts table loads
- [ ] Alert history graph renders

- [ ] Alert rules status shows

5. Real-Time Updates

```bash
# Trigger some activity
curl -X POST http://localhost:8001/api/v1/analyze
curl -X POST http://localhost:8002/api/v1/optimize

# Wait 10 seconds, check dashboard
# Graphs should update automatically

# Verify auto-refresh is working
# Look for "Last updated" timestamp in Grafana UI
```

6. Variables Working

In any dashboard:

- [ ] Click time range selector - options appear
- [ ] Change time range - data updates
- [ ] Click refresh dropdown - intervals shown
- [ ] Change refresh rate - updates occur
- [ ] If variable dropdowns present - they work

7. Run Tests

```bash
cd monitoring/tests
pytest test_grafana.py -v
# Expected: All tests pass (6/6)
```

🐛 Troubleshooting

Issue 1: Grafana Won't Start

Symptoms: Container exits immediately or won't start

Solutions:

```bash
```

```
# Check Grafana logs
docker logs optiinfra-grafana

# Common issues:
# 1. Port 3000 already in use
sudo lsof -i :3000
# Kill the process or change port

# 2. Permission issues with volumes
sudo chown -R 472:472 monitoring/grafana/
# 472 is the grafana user ID

# 3. Invalid grafana.ini
# Validate syntax, remove any invalid options

# 4. Missing provisioning files
ls -la monitoring/grafana/provisioning/
# Ensure all directories exist
```

Issue 2: Prometheus Data Source Not Working

Symptoms: "Data source connected" but no data in panels

Solutions:

```
bash

# Test Prometheus URL from Grafana container
docker exec optiinfra-grafana curl http://prometheus:9090/api/v1/query?query=up

# Check data source configuration
curl -u admin:optiinfra_admin http://localhost:3000/api/datasources
# Verify URL is http://prometheus:9090

# Test query directly in Grafana
# Go to Explore tab
# Run: up
# Should return results

# Check Prometheus has data
curl http://localhost:9090/api/v1/query?query=up
```

Issue 3: Dashboards Not Loading

Symptoms: Dashboard list is empty or dashboards missing

Solutions:

```bash
# Check provisioning configuration
cat monitoring/grafana/provisioning/dashboards/optiinfra.yml
# Verify path is correct

# Check dashboard JSON files exist
ls -la monitoring/grafana/dashboards/
# Should see 7 .json files

# Check Grafana logs for provisioning errors
docker logs optiinfra-grafana | grep -i provision

# Force reload dashboards
docker restart optiinfra-grafana

# Manual dashboard import (if provisioning fails)
# In Grafana UI: + → Import → Upload JSON file
```

Issue 4: Panels Show "No Data"

Symptoms: Dashboard loads but panels empty

Solutions:

```bash
```

```
# Check if Prometheus has metrics
curl http://localhost:9090/api/v1/query?query=agent_requests_total
# Should return data

# Check time range
# Set to "Last 5 minutes" and ensure services have been running

# Generate some test data
curl -X POST http://localhost:8001/api/v1/health
curl -X POST http://localhost:8002/api/v1/health

# Wait 15 seconds for scrape, refresh dashboard

# Check PromQL query in panel
# Click panel title → Edit
# Run query in Prometheus UI first to validate

# Check datasource variable
# Ensure $datasource resolves to Prometheus
```

Issue 5: Dashboard Variables Not Working

Symptoms: Dropdown variables show "No options" or don't filter

Solutions:

```
json
```

```
// Check variable query in dashboard JSON
// Example variable definition:
{
  "name": "service",
  "type": "query",
  "datasource": "${DS_PROMETHEUS}",
  "query": "label_values(up, job)",
  "refresh": 1
}

// Test variable query in Prometheus
// label_values(up, job) should return service names

// Check variable is used in panel queries
// Should see: {job="$service"}

// Refresh variables manually
// Dashboard settings → Variables → Refresh button
```

Issue 6: Auto-Refresh Not Working

Symptoms: Data doesn't update automatically

Solutions:

```bash
# Check refresh rate is set
# Top right corner should show refresh interval (5s, 15s, etc.)

# Verify Prometheus is scraping
curl http://localhost:9090/api/v1/targets
# All targets should be "up"

# Check browser console for errors
# F12 → Console tab

# Try manual refresh
# Click refresh button in top right

# Check panel query interval
# Edit panel → Query options → Min interval
# Should be empty or <= refresh rate
```

📊 Dashboard Panel Examples

Stat Panel (Cost Savings)

json

```
{
  "type": "stat",
  "title": "Total Cost Savings (24h)",
  "targets": [{
    "expr": "sum(increase(cost_savings_total[24h]))",
    "legendFormat": "Savings"
  }],
  "options": {
    "graphMode": "area",
    "colorMode": "value",
    "orientation": "horizontal",
    "reduceOptions": {
      "values": false,
      "fields": "",
      "calcs": ["lastNotNull"]
    }
  },
  "fieldConfig": {
    "defaults": {
      "unit": "currencyUSD",
      "color": {"mode": "thresholds"},
      "thresholds": {
        "steps": [
          {"value": 0, "color": "red"},
          {"value": 1000, "color": "yellow"},
          {"value": 5000, "color": "green"}
        ]
      }
    }
  }
}
```

Time Series Graph (Request Rate)

json

```
{
  "type": "timeseries",
  "title": "Request Rate by Agent",
  "targets": [{
    "expr": "sum by (agent) (rate(agent_requests_total[5m]))",
    "legendFormat": "{{agent}}"
  }],
  "options": {
    "tooltip": {"mode": "multi"},
    "legend": {
      "displayMode": "table",
      "placement": "right",
      "calcs": ["mean", "max", "last"]
    }
  },
  "fieldConfig": {
    "defaults": {
      "unit": "reqps",
      "custom": {
        "drawStyle": "line",
        "lineWidth": 2,
        "fillOpacity": 10,
        "showPoints": "never"
      }
    }
  }
}
```

Gauge (GPU Utilization)

```
json
```

```
{
  "type": "gauge",
  "title": "Average GPU Utilization",
  "targets": [{
    "expr": "avg(gpu_utilization)",
    "legendFormat": "GPU"
  }],
  "options": {
    "showThresholdLabels": true,
    "showThresholdMarkers": true
  },
  "fieldConfig": {
    "defaults": {
      "unit": "percent",
      "min": 0,
      "max": 100,
      "thresholds": {
        "steps": [
          {"value": 0, "color": "red"},
          {"value": 50, "color": "yellow"},
          {"value": 70, "color": "green"}
        ]
      }
    }
  }
}
```

Heatmap (GPU Utilization by ID)

```
json
```

```
{
  "type": "heatmap",
  "title": "GPU Utilization Heatmap",
  "targets": [{
    "expr": "gpu_utilization",
    "legendFormat": "{{gpu_id}}"
  }],
  "options": {
    "calculate": false,
    "cellGap": 2,
    "cellRadius": 0,
    "color": {
      "scheme": "Spectral",
      "exponent": 0.5,
      "min": 0,
      "max": 100
    }
  },
  "fieldConfig": {
    "defaults": {
      "unit": "percent"
    }
  }
}
```

Table (Recent Recommendations)

```
json
```

```
{
  "type": "table",
  "title": "Recent Recommendations",
  "targets": [{
    "expr": "topk(10, cost_recommendations_total)",
    "legendFormat": "",
    "format": "table",
    "instant": true
  }],
  "options": {
    "showHeader": true,
    "sortBy": [{
      "displayName": "Time",
      "desc": true
    }]
  },
  "fieldConfig": {
    "defaults": {},
    "overrides": [
      {
        "matcher": {"id": "byName", "options": "type"},
        "properties": [{
          "id": "custom.width",
          "value": 150
        }]
      }
    ]
  }
}
```

🎨 Color Scheme & Styling

OptiInfra Color Palette

```
css
```

```
/* Success / Positive */
--success-green: #00C853;
--light-green: #00E676;

/* Error / Negative */
--error-red: #F44336;
--dark-red: #D32F2F;

/* Warning */
--warning-orange: #FF9800;
--light-orange: #FFB74D;

/* Info / Neutral */
--info-blue: #2196F3;
--light-blue: #64B5F6;

/* Background */
--bg-dark: #0a0a0a;
--bg-medium: #1E3A5F;
--bg-light: #2C5282;

/* Text */
--text-primary: #FFFFFF;
--text-secondary: rgba(255,255,255,0.9);
--text-tertiary: rgba(255,255,255,0.7);
```

Threshold Configuration

Cost Savings:

- 0-$1,000: Red (poor)
- $1,000-$5,000: Yellow (moderate)
- $5,000+: Green (excellent)

Error Rate:

- 0-1%: Green (excellent)
- 1-5%: Yellow (acceptable)
- 5%+: Red (critical)

Latency:

- 0-100ms: Green (excellent)
- 100-500ms: Yellow (acceptable)
- 500ms+: Red (slow)

GPU Utilization:

- 0-50%: Red (underutilized)
- 50-70%: Yellow (moderate)
- 70-95%: Green (optimal)
- 95%+: Orange (risk of saturation)

Quality Score:

- 0.95-1.0: Green (excellent)
- 0.90-0.95: Yellow (acceptable)
- <0.90: Red (poor)

🔗 Dashboard Links & Navigation

Create links between dashboards for easy navigation:

Overview Dashboard Links

```json
{
  "links": [
    {"title": "Cost Agent", "url": "/d/optiinfra-cost"},
    {"title": "Performance Agent", "url": "/d/optiinfra-performance"},
    {"title": "Resource Agent", "url": "/d/optiinfra-resource"},
    {"title": "Application Agent", "url": "/d/optiinfra-application"},
    {"title": "Infrastructure", "url": "/d/optiinfra-infra"},
    {"title": "Alerts", "url": "/d/optiinfra-alerts"}
  ]
}
```

Panel Drill-Down Links

Add data links to panels:

json

```
{
  "fieldConfig": {
    "defaults": {
      "links": [{
        "title": "View in Cost Agent Dashboard",
        "url": "/d/optiinfra-cost?var-service=${__field.labels.service}"
      }]
    }
  }
}
```

🔔 Alert Notification Setup

Slack Notification Channel

yaml

```
# monitoring/grafana/provisioning/notifiers/slack.yml
notifiers:
  - name: slack-optiinfra
    type: slack
    uid: slack1
    org_id: 1
    is_default: true
    settings:
      url: https://hooks.slack.com/services/YOUR/WEBHOOK/URL
      recipient: "#optiinfra-alerts"
      username: OptiInfra Grafana
      icon_emoji: ":rocket:"
      mentionChannel: "here"
    secure_settings:
      url: ${SLACK_WEBHOOK_URL}  # From environment variable
```

Email Notification Channel

yaml

```yaml
# monitoring/grafana/provisioning/notifiers/email.yml
notifiers:
  - name: email-optiinfra
    type: email
    uid: email1
    org_id: 1
    settings:
      addresses: "team@optiinfra.com;oncall@optiinfra.com"
      singleEmail: false
```

📈 Advanced PromQL Queries

Complex Aggregations

promql

```promql
# Cost savings rate with trend
deriv(sum(increase(cost_savings_total[1h]))[10m:1m])

# Error budget consumption (SLO: 99.9% uptime)
1 - ((sum(rate(requests_total[30d])) - sum(rate(errors_total[30d]))) / sum(rate(requests_total[30d])))

# Apdex score (Application Performance Index)
(sum(rate(request_duration_seconds_bucket{le="0.1"}[5m])) +
 sum(rate(request_duration_seconds_bucket{le="0.5"}[5m])) / 2) /
sum(rate(request_duration_seconds_count[5m]))

# GPU utilization efficiency (workload vs capacity)
avg(gpu_utilization) /
(count(gpu_utilization) * 100)

# Cost per optimization
sum(increase(cost_total[24h])) /
sum(increase(optimization_executions_total[24h]))
```

Predictions & Forecasting

promql

```
# Predict cost savings in next hour (linear regression)
predict_linear(cost_savings_total[6h], 3600)

# Predict when GPU will reach 90% utilization
predict_linear(avg(gpu_utilization)[30m],
  (90 - avg(gpu_utilization)) / deriv(avg(gpu_utilization)[30m]))
```

📦 Export & Backup

Export Dashboard JSON

bash

```
# Export single dashboard
curl -u admin:optiinfra_admin \
  http://localhost:3000/api/dashboards/uid/optiinfra-overview | \
  jq '.dashboard' > overview-backup.json

# Export all dashboards
for dash in $(curl -s -u admin:optiinfra_admin http://localhost:3000/api/search | jq -r '.[].uid'); do
  curl -s -u admin:optiinfra_admin \
    http://localhost:3000/api/dashboards/uid/$dash | \
    jq '.dashboard' > monitoring/grafana/dashboards/backup-$dash.json
done
```

Import Dashboard

bash

```
# Via API
curl -X POST -u admin:optiinfra_admin \
  -H "Content-Type: application/json" \
  -d @dashboard.json \
  http://localhost:3000/api/dashboards/db

# Via UI
# + → Import → Upload JSON file
```

🔄 Integration with Next Steps

This prompt (0.11 PART 2) completes the Foundation phase. Next:

1. **DECISION GATE 2** (End of Week 1)
   - Verify all infrastructure operational
   - All dashboards showing data
   - Ready to proceed to agents
2. **PHASE 1: COST AGENT** (Week 2-3)
   - Use Cost Agent dashboard to track development
   - Monitor savings in real-time
   - Validate optimizations via Grafana
3. **PHASE 5: PRODUCTION** (Week 10)
   - Add customer-specific dashboards
   - Setup alert escalation (PagerDuty)
   - Create public status page

📁 Generated Files Checklist

After running this prompt, you should have:

- [ ] `monitoring/grafana/grafana.ini` - Grafana configuration
- [ ] `monitoring/grafana/provisioning/datasources/prometheus.yml` - Data source
- [ ] `monitoring/grafana/provisioning/dashboards/optiinfra.yml` - Dashboard provider
- [ ] `monitoring/grafana/provisioning/notifiers/slack.yml` - Slack notifications
- [ ] `monitoring/grafana/dashboards/overview.json` - Overview dashboard
- [ ] `monitoring/grafana/dashboards/cost-agent.json` - Cost Agent dashboard
- [ ] `monitoring/grafana/dashboards/performance-agent.json` - Performance dashboard
- [ ] `monitoring/grafana/dashboards/resource-agent.json` - Resource dashboard
- [ ] `monitoring/grafana/dashboards/application-agent.json` - Application dashboard
- [ ] `monitoring/grafana/dashboards/infrastructure.json` - Infrastructure dashboard
- [ ] `monitoring/grafana/dashboards/alerts.json` - Alerts dashboard
- [ ] `monitoring/tests/test_grafana.py` - Grafana tests
- [ ] `docker-compose.yml` - Updated with Grafana service
- [ ] `monitoring/README-grafana.md` - Grafana setup guide

⏱️ Time Breakdown

Task	Estimated Time
Grafana config & provisioning	5 minutes
Overview dashboard creation	10 minutes
Agent dashboards (4x)	40 minutes
Infrastructure dashboard	8 minutes
Alerts dashboard	5 minutes
Docker Compose updates	2 minutes
Testing and validation	10 minutes
TOTAL	**80 minutes**

Actual time: 20 minutes (Windsurf) + 10 minutes (validation) = **30 minutes**

📚 Additional Resources

- Grafana Documentation
- Dashboard Best Practices
- PromQL in Grafana
- Dashboard JSON Schema
- Provisioning Dashboards

✅ Completion Checklist

Mark when complete:

- [ ] Grafana server running and accessible
- [ ] Prometheus data source connected and working
- [ ] All 7 dashboards loaded via provisioning
- [ ] Overview dashboard shows all services
- [ ] Cost Agent dashboard shows metrics
- [ ] Performance Agent dashboard operational
- [ ] Resource Agent dashboard displays GPU metrics
- [ ] Application Agent dashboard shows quality scores

- [ ] Infrastructure dashboard shows all databases
- [ ] Alerts dashboard shows alert status
- [ ] Auto-refresh working (5s-30s intervals)
- [ ] Time range selector functional
- [ ] Dashboard variables working
- [ ] All tests passing (6/6)
- [ ] Can navigate between dashboards via links
- [ ] Notification channels configured (optional)
- [ ] Dashboard JSONs exported to git
- [ ] Documentation complete
- [ ] Ready for DECISION GATE 2

Document Version: 1.0
Last Updated: October 21, 2025
Previous: FOUNDATION-0.11 PART 1 (Prometheus Setup)
Next: DECISION GATE 2 (Foundation Complete)
Estimated Completion: 30 minutes